Linköping University
Department of Computer and Information Science (IDA)
The UPP group 2019-08-29

Short Valgrind Guide

Valgrind is a collection of tools that we can use for analysis of our programs. In this document we will see how we use Valgrind to find memory leaks (If you want to know more about Valgrind you can visit their website, www.valgrind.org).

Step 1: Compilation

In the file `test.cc` we have written a program that we want to examine for memory leaks. When vi compile this program we have to add the `-g` flag. This is done so that Valgrind will have all the necessary information for it to find any memory leaks. In every other aspect compilation works the same way:

```
g++ -std=c++17 -g test.cc
```

(NOTE: `test.cc` is just an example file, it should be replaced with your file(s).)

Step 2: Running Valgrind

We have now compiled all files thus producing an executable file (`a.out` if nothing else have been specified).

To examine the program for memory leaks you now have to run this executable file through Valgrind. This is done like this:

```
valgrind --tool=memcheck --leak-check=yes ./a.out
```

Where `a.out` is the name of the executable file and `./` tells Valgrind that the exectuable file is located in the current directory.

Valgrind will now run your program and print a lot of information about it. Now we just have to understand the output.

Step 3.a: Memory leaks are present

The following program is an example where there are memory leaks. On line 7 in the program, under `main`, we have allocated memory for an object of type `A`. This memory is never deallocated.

test.cc

```
1 class A{
2   int x;
3 };
4
5 int main()
6 {
7    A * a = new A();
8    return 0;
9 }
```

After we have compiled the program and run it through Valgrind the following (or something very similar) is printed:

```
==18420== Memcheck, a memory error detector
==18420== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==18420== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==18420== Command: ./a.out
==18420==
==18420==
==18420== HEAP SUMMARY:
==18420==     in use at exit: 4 bytes in 1 blocks
==18420==   total heap usage: 2 allocs, 1 frees, 72,708 bytes allocated
==18420==
==18420== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==18420==    at 0x4C3017F: operator new(unsigned long)
==18420==    by 0x10868B: main (test.cc:8)
==18420==
==18420== LEAK SUMMARY:
==18420==    definitely lost: 4 bytes in 1 blocks
==18420==    indirectly lost: 0 bytes in 0 blocks
==18420==      possibly lost: 0 bytes in 0 blocks
==18420==    still reachable: 0 bytes in 0 blocks
==18420==         suppressed: 0 bytes in 0 blocks
==18420==
==18420== For counts of detected and suppressed errors, rerun with: -v
==18420== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

In this wall-of-text Valgrind tells us two important things:

1. On line 8, in the file `test.cc`, in the function `main`, there is memory allocated that never gets deallocated. This is specified by
   ```
   ==18420==    by 0x10868B: main (test.cc:8)
   ```
2. That we have lost 4 bytes of data, as stated i `LEAK SUMMARY`:
   ```
   ==18420== LEAK SUMMARY:
   ==18420==    definitely lost: 4 bytes in 1 blocks
   ==18420==    indirectly lost: 0 bytes in 0 blocks
   ==18420==      possibly lost: 0 bytes in 0 blocks
   ==18420==    still reachable: 0 bytes in 0 blocks
   ==18420==         suppressed: 0 bytes in 0 blocks
   ```

Step 3.b: No memory leaks are present

The following is an example of a program where there are no memory leaks.

test.cc

```
class A{
  int x;
};

int main()
{
  A * a = new A();
  delete a;
  return 0;
}
```

After we have compiled the program and run it through Valgrind we get the following output:

```
==13369== Memcheck, a memory error detector
==13369== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13369== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==13369== Command: ./a.out
==13369==
==13369==
==13369== HEAP SUMMARY:
==13369==     in use at exit: 0 bytes in 0 blocks
==13369==   total heap usage: 2 allocs, 2 frees, 72,708 bytes allocated
==13369==
==13369== All heap blocks were freed -- no leaks are possible
==13369==
==13369== For counts of detected and suppressed errors, rerun with: -v
==13369== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

If there are no memory leaks Valgrind will print the line:

```
==13369== All heap blocks were freed -- no leaks are possible
```